1 Randomisierte Bestimmung des Medians

1.1 Problemstellung und Ziel

In diesem Abschnitt stellen wir einen randomisierten Algorithmus zur Bestimmung des Medians vor, der besser[1] ist als jeder theoretisch mögliche deterministische Algorithmus. Unter dem Median versteht man das „mittlere" Element einer sortierten Sequenz von Elementen aus einer vollständig geordneten Menge (beispielsweise die natürlichen Zahlen). Dies sieht man leicht an folgendem Beispiel:

Gegeben sei die Menge $S := \{7, 3, 4, 8, 1\}$, bzw. $(1, 3, 4, 7, 8)$ in sortierter Reihenfolge. Der Median von S ist „4". Beachte: Der Median ist nicht gleich dem Mittelwert einer Sequenz (im Beispiel: $\frac{1}{5}(7 + 3 + 4 + 8 + 1) = \frac{23}{5}$). Da der Median einer geraden Anzahl von Elementen nicht eindeutig festgelegt ist, definieren wir den Median als das $\lceil \frac{n}{2} \rceil$-te Element der sortierten Sequenz.

Wir stellen uns im Folgenden vor, daß die Eingabemenge S als Array $a[1], \dots, a[n]$ vorliegt, wobei das Array selbstverständlich nicht sortiert sein muß (sonst ist die Aufgabe trivial). Der gesuchte Algorithmus soll den Median dieses Arrays bestimmen. Dies könnte man beispielsweise dadurch realisieren, daß man alle Elemente von $a[.]$ sortiert. Hierfür wird allerdings Zeit $\Theta(n \log n)$ benötigt. Wir streben jedoch einen Algorithmus mit Laufzeit $\mathcal{O}(n)$ an.

1.2 Schema des Algorithmus

Die Lösungsidee besteht darin, m Elemente von $a[1], \dots, a[n]$ zufällig auszuwählen (unabhängig und jeweils[2] mit Wahrscheinlichkeit $\frac{1}{n}$) und in ein Array $b[1], \dots, b[m]$ einzutragen. Dabei sei m deutlich kleiner als n. Konkret werden wir $m = n^{\frac{3}{4}}$ setzen. Das Teilarray $b[.]$ sortieren wir in Zeit $\mathcal{O}(n^{\frac{3}{4}} \log n^{\frac{3}{4}}) = o(n)$.

Wegen der unabhängigen Auswahl der Elemente erwarten wir, daß der Median von $b[.]$ (das Element $b[\lceil \frac{1}{2} n^{\frac{3}{4}} \rceil]$ nach dem Sortieren) ungefähr gleich dem Median von $a[.]$ ist. Dieses Wissen verwenden wir, um eine kleine Menge von Kandidaten für den Median von $a[.]$ zu erhalten. Dazu betrachten wir zwei Pivotelemente

$$p_\ell = b[\lfloor \tfrac{1}{2} n^{\frac{3}{4}} - \sqrt{n} \rfloor] \quad \text{und} \quad p_r = b[\lceil \tfrac{1}{2} n^{\frac{3}{4}} + \sqrt{n} \rceil]$$

Wir werden zeigen, daß die Menge $T := \{x \in S \ : \ p_\ell \le x \le p_r\}$ aller Elemente von $a[.]$, die zwischen p_ℓ und p_r liegen, mit hoher Wahrscheinlichkeit die folgenden Eigenschaften besitzt:

1. Der Median von $a[.]$ liegt in T.

2. T ist relativ klein, konkret $|T| < 4n^{\frac{3}{4}}$.

Abbildung 1.1 veranschaulicht unser Vorgehen.

[1] „Besser" bezieht sich hierbei auf die Anzahl benötigter Vergleiche.

[2] Wir wählen also eventuell manche Elemente mehrfach, aber da dies nur mit geringer Wahrscheinlichkeit passieren wird, soll uns dies nicht weiter stören.

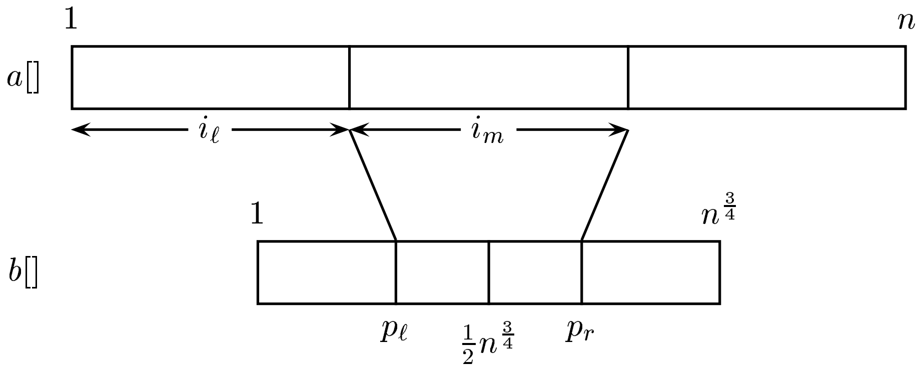

Abbildung 1.1: Randomisierte Bestimmung des Medians

Da T klein ist, können wir diese Menge in Zeit $\mathcal{O}(n^{\frac{3}{4}} \log n^{\frac{3}{4}}) = o(n)$ sortieren. Daraus erhalten wir den gesuchten Median wie folgt: Mit i_ℓ und $(i_\ell + i_m)$ bezeichnen wir den *Rang* von p_ℓ bzw. p_r Unter dem Rang verstehen wir den Index eines Elements in der sortierten Liste. Im obigen Beispiel hat das Element 7 also den Rang 4. Aus i_ℓ können wir leicht berechnen, wo der gesuchte Median in der sortierten Menge T liegt. Algorithmus 1.1 faßt dieses Vorgehen zusammen.

Algorithmus 1.1 funktioniert allerdings nur unter der Bedingung, daß die Eingabe $a[1], \ldots, a[n]$ keine Duplikate enthält (*Voraussetzung A*). Der Leser überlege sich, was passiert, wenn beispielsweise $a[1] = \ldots = a[n]$ gilt.

Unter Voraussetzung A ist die Korrektheit von Algorithmus 1.1 relativ leicht einzusehen und wir überlassen dies deshalb dem Leser. Im Grunde muß man sich hierbei nur klarmachen, daß am Ende $b[\lceil \frac{n}{2} \rceil - i_\ell + 1]$ tatsächlich den Median enthält. Das folgende Theorem zeigt, daß der Algorithmus mit hoher Wahrscheinlichkeit nur Laufzeit $\mathcal{O}(n)$ benötigt.

Satz 1 *Mit Wahrscheinlichkeit* $1 - n^{-\frac{1}{4}}$ *findet Algorithmus 1.1 den Median in Laufzeit* $\mathcal{O}(n)$, *wobei höchstens* $\frac{3}{2}n + o(n)$ *Vergleiche durchgeführt werden.*

Beweis: Wir betrachten im Folgenden eine Iteration der repeat-Schleife. Man sieht relativ leicht, daß die Partitionierung von $a[.]$ die Laufzeit der Iteration dominiert und wir deshalb insgesamt mit Zeit $\mathcal{O}(n)$ pro Schleifendurchlauf auskommen.

Bei den beiden Aufrufen eines Sortieralgorithmus werden jeweils $o(n)$ Vergleiche benötigt (wie oben dargestellt). Wir müssen also nur noch die Vergleiche beim Partitionieren betrachten. Man sieht sofort, daß dafür $2n$ Vergleiche ausreichen. Wenn man jedoch bei der Berechnung von $p_\ell \leq a[j] \leq p_r$ den ersten durchzuführenden Vergleich zufällig auswählt, so wird für alle Elemente von $a[.]$, die nicht in T liegen[3], der zweite Vergleich mit Wahrscheinlichkeit $\frac{1}{2}$ hinfällig, da entweder $a[j] < p_\ell$ oder $a[j] > p_r$ gilt. Dies rechnen wir aber nicht formal nach, sondern überlassen dies dem Leser als Übungsaufgabe.

[3]Und das sind fast alle, genauer gesagt $n - 4n^{\frac{3}{4}} = (1 - o(1))n$ viele.

Es genügt nun, zu zeigen, daß der Algorithmus mit Wahrscheinlichkeit $1 - n^{-\frac{1}{4}}$ nach nur einer Iteration terminiert. Dazu zeigt man im einzelnen:

1. $\Pr[i_\ell > \lceil \frac{n}{2} \rceil] \leq \frac{1}{4} n^{-\frac{1}{4}}$ (linke Grenze beim Partitionieren zu groß)
2. $\Pr[i_\ell + i_m < \lceil \frac{n}{2} \rceil] \leq \frac{1}{4} n^{-\frac{1}{4}}$ (rechte Grenze beim Partitionieren zu klein)
3. $\Pr[i_m \geq 4n^{\frac{3}{4}}] \leq \frac{1}{2} n^{-\frac{1}{4}}$ (zu viele Elemente liegen in T)

Wir rechnen im Folgenden nur die erste Behauptung nach. Die anderen Aussagen folgen mit derselben Technik und wir stellen sie wieder als Übungsaufgabe.

Sei $S_<$ die Menge der kleinsten $\lceil \frac{n}{2} \rceil$ Elemente von $a[.]$ (alle kleinen Elemente einschließlich des Medians). Wenn wir bei der zufälligen Auswahl höchstens $\frac{1}{2} n^{\frac{3}{4}} - \sqrt{n}$ Elemente aus $S_<$ selektieren[4], so gehört p_ℓ nicht zu $S_<$. Dann und nur dann erhalten wir $i_\ell > \lceil \frac{n}{2} \rceil$.

Wir verwenden die Zufallsvariablen

$$X_i := \begin{cases} 1 & \text{falls } b[i] \text{ zu } S_< \text{ gehört} \\ 0 & \text{sonst} \end{cases}$$

Dann gilt $\Pr[X_i = 1] = \frac{1}{2}$ und $X := \sum_{i=1}^{n^{\frac{3}{4}}} X_i$ ist binomialverteilt. Daraus folgt

$$\mathbb{E}[X] = \frac{1}{2} n^{\frac{3}{4}} \quad \text{und} \quad \mathrm{Var}[X] = \frac{1}{4} n^{\frac{3}{4}}$$

Die gesuchte Wahrscheinlichkeit schätzen wir mit der Ungleichung von Chebyshev ab:

$$\begin{aligned} \Pr\left[i_\ell > \left\lceil \frac{n}{2} \right\rceil\right] &\leq \Pr\Big[X \leq \underbrace{\frac{1}{2} n^{\frac{3}{4}}}_{\mathbb{E}[X]} - \sqrt{n}\Big] = \Pr[X - \mathbb{E}[X] \leq -\sqrt{n}] \\ &\leq \Pr[|X - \mathbb{E}[X]| \geq \sqrt{n}] \leq \frac{\mathrm{Var}[X]}{n} = \frac{1}{4} n^{-\frac{1}{4}} \end{aligned}$$

Die Gültigkeit der Eigenschaften 2. und 3. zeigt man analog. □

Bemerkung 2 *Die Tatsache, daß man mit sehr hoher Wahrscheinlichkeit mit* $\frac{3}{2}n + o(n)$ *vielen Vergleichen auskommt, ist insbesondere deshalb sehr interessant, da man zeigen kann, daß jeder* deterministische *Algorithmus mehr als* $2n$ *Vergleiche braucht, um den Median zu finden. Der beste bislang bekannte deterministische Medianalgorithmus benötigt bereits* $2.95n$ *Vergleiche und ist deutlich komplizierter als der von uns betrachtete randomisierte Algorithmus.*

[4] Wir vernachlässigen hier, daß eventuell noch gerundet werden muß, und daß als Index für das Element p_ℓ in $b[.]$ keine Zahl kleiner Eins verwendet werden darf. Dies wird im Algorithmus durch die Maximumsbildung sichergestellt.

1.3 Hinweise zur Implementierung

1.3.1 Durchführung der Vergleiche

Zwar ist es aus theoretischer Sicht interessant, daß man die Anzahl der Vergleich wie im Beweis von Satz 1 beschrieben auf $\frac{3}{2} + o(n)$ beschränken kann, wenn man den ersten auszuführenden Vergleich zufällig wählt. Bei einer praktischen Implementierung kann man jedoch auf diese Feinheit verzichten, da vermutlich die Berechnung einer Zufallszahl mehr Zeit in Anspruch nimmt als der zusätzliche Vergleich.

1.3.2 Behandlung von Duplikaten

Wie bereits zuvor erwähnt wurde bei der Darstellung des Algorithmus im vorigen Abschnitt etwas „geschummelt", da das Verfahren nur unter Voraussetzung A (keine Duplikate) korrekt ist. Allerdings kann man dieses Problem durch eine kleine Modifikation des Algorithmus umgehen.

Beispielsweise ist folgendes Vorgehen denkbar: Man unterscheidet bei der Partitionierung nicht nur die Fälle $a[j] < p_\ell$, $p_\ell \leq a[j] \leq p_r$ und $p_r < a[j]$, sondern auch noch $a[j] = p_\ell$ und $a[j] = p_r$. Das heißt, man unterteilt das Array $a[.]$ in fünf Bereiche statt wie bisher in drei. Durch Mitführen geeigneter Zähler kann festgestellt werden, wo sich diese Bereiche erstrecken (wie bisher mittels i_ℓ und i_m). Wenn also z. B. weniger als $\lceil \frac{n}{2} \rceil$ Elemente echt kleiner als p_ℓ sind, aber mindestens $\lceil \frac{n}{2} \rceil$ Elemente kleiner gleich p_ℓ, so wissen wir, daß der Median gleich p_ℓ ist.

1.3.3 Selektion des k-ten Elements

Algorithmus 1.1 kann direkt in einen Algorithmus zur Selektion des k-ten Elements (statt des $\lceil \frac{n}{2} \rceil$-ten Elements) umgewandelt werden. Dazu muß man einfach statt $\frac{1}{2}n^{\frac{3}{4}}$ den Wert $\frac{k}{n}n^{\frac{3}{4}} = kn^{-\frac{1}{4}}$ verwenden. Ferner wird bei der Überprüfung der Indizes i_ℓ und i_m, ob das gesuchte Element zwischen p_ℓ und p_m liegt, in der allgemeinen Version mit k statt mit $\lceil \frac{n}{2} \rceil$ verglichen.

Algorithmus 1.1 Randomisierte Bestimmung des Medians

func randMedian(a[.])

repeat
  // *Wähle* $n^{\frac{3}{4}}$ *Elemente aus* a[.] *zufällig aus*
  for $(i = 1;\ i \leq \lceil n^{\frac{3}{4}} \rceil;\ i++)$ {
    $b[i] = a[\text{RANDOM}(1, n)];$
  }
  Sortiere die Elemente in $b[]$ (mit einem $\mathcal{O}(n \log n)$ Sortieralgorithmus);
  $p_\ell = b[\max\{\lfloor \frac{1}{2} n^{\frac{3}{4}} - \sqrt{n} \rfloor, 1\}];$
  $p_r = b[\min\{\lceil \frac{1}{2} n^{\frac{3}{4}} + \sqrt{n} \rceil, \lceil n^{\frac{3}{4}} \rceil\}];$
  // *Bestimme den Rang von* p_ℓ *und* p_r *und trage* T *in* b[.] *ein*
  $i_\ell = i_m = 0;$
  for $(j = 1;\ j \leq n;\ j++)$ {
    if $p_\ell \leq a[j] \leq p_r$ **then** $i_m++;\ b[i_m] = a[j];$
    else if $a[j] < p_\ell$ **then** $i_\ell++;$
  }
until (
  // *Median in* T
  $i_\ell \leq \lceil \frac{n}{2} \rceil$ **and** $i_\ell + i_m \leq \lceil \frac{n}{2} \rceil$ **and**
  // *T nicht zu groß*
  $i_m < 4n^{\frac{3}{4}}$)
Sortiere b[.] und gib den Median $b[\lceil \frac{n}{2} \rceil - i_\ell + 1]$ zurück.

1.4 Eine einfache Alternative

In der Praxis verwendet man häufig zur Median-Bestimmung einen anderen Algorithmus, der sehr einfach zu implementieren ist. Es handelt sich dabei um eine Variante der QuickSort-Strategie:

- Wähle ein Pivot-Element (am besten zufällig, evtl. sogar in Kombination mit der Median-of-Three-Strategie).
- Partitioniere das Array.
- Wende den Algorithmus rekursiv auf die Partition an, in der sich der Median (bzw. das k-te Element) befindet. Dies ist problemlos möglich, da man beim Partitionieren leicht den Rang des Pivot-Elements ermitteln kann. Wenn das Pivot-Element schon den gesuchten Rang hat, so wird es natürlich ohne weitere rekursive Aufrufe zurückgeliefert.

Dieser sogenannte QuickSelect-Algorithmus hat also exakt dieselbe Struktur wie QuickSort. Der einzige wesentliche Unterschied besteht darin, daß nur ein rekursiver Aufruf statt zwei durchgeführt wird.